

# Package ‘VariantAnnotation’

March 26, 2013

**Type** Package

**Title** Annotation of Genetic Variants

**Description** Annotate variants with respect to location and amino acid coding

**Version** 1.4.12

**Author** Valerie Obenchain, Martin Morgan, Michael Lawrence

**Maintainer** Valerie Obenchain <vobencha@fhcrc.org>

**License** Artistic-2.0

**biocViews** DataImport, Sequencing, HighThroughputSequencing, SNP,Annotation, Genetics, Homo\_sapiens

**Depends** R (>= 2.8.0), methods, GenomicRanges (>= 1.9.56), Rsamtools (>= 1.9.4), IRanges (>= 1.15.48)

**Imports** methods, BiocGenerics (>= 0.1.0), IRanges, Biobase (>= 2.15.1), Rsamtools (>= 1.9.4), AnnotationDbi (>= 1.17.11), Biostrings (>= 2.25.2), zlibbioc, BSgenome, GenomicFeatures (>= 1.9.35), DBI

**Suggests**

RUnit, BSgenome.Hsapiens.UCSC.hg19, TxDb.Hsapiens.UCSC.hg19.knownGene, SNPlocs.Hsapiens.dbSNP.2011081

**LinkingTo** IRanges, Biostrings, Rsamtools

**LazyLoad** yes

## R topics documented:

getTranscriptSeqs . . . . .	2
locateVariants . . . . .	3
MatrixToSnpMatrix . . . . .	6
PolyPhenDb-class . . . . .	8
PolyPhenDbColumns . . . . .	10
predictCoding . . . . .	12
readVcf . . . . .	15
readVcfLongForm . . . . .	18
refLocsToLocalLocs . . . . .	20
scanVcf . . . . .	22

ScanVcfParam-class . . . . .	24
SIFTDb-class . . . . .	26
SIFTDbColumns . . . . .	28
summarizeVariants . . . . .	29
VAFilter-class . . . . .	31
VAFilterResult-class . . . . .	33
VariantType-class . . . . .	35
VAUtil-class . . . . .	36
VCF-class . . . . .	36
VCFHeader-class . . . . .	39
writeVcf . . . . .	41
<b>Index</b>	<b>43</b>

---

getTranscriptSeqs	<i>Get transcript sequences</i>
-------------------	---------------------------------

---

## Description

Extract transcript sequences from a [BSgenome](#) object or an [FaFile](#).

## Usage

```
## S4 method for signature 'GRangesList,BSgenome'
getTranscriptSeqs(query, subject, ...)
## S4 method for signature 'GRangesList,FaFile'
getTranscriptSeqs(query, subject, ...)
## S4 method for signature 'GRanges,FaFile'
getTranscriptSeqs(query, subject, ...)
```

## Arguments

query	A <a href="#">GRangesList</a> object containing exons or cds grouped by transcript.
subject	A <a href="#">BSgenome</a> object or a <a href="#">FaFile</a> from which the sequences will be taken.
...	Additional arguments

## Details

getTranscriptSeqs is a wrapper for the extractTranscriptsFromGenome and getSeq functions. The purpose is to allow sequence extraction from either a [BSgenome](#) or [FaFile](#). Transcript sequences are extracted based on the boundaries of the feature provided in the query (i.e., either exons or cds regions).

## Value

A [DNAStringSet](#) instance containing the sequences for all transcripts specified in query.

## Author(s)

Valerie Obenchain

**See Also**

[predictCoding](#) [extractTranscriptsFromGenome](#) [getSeq](#)

**Examples**

```
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
library(BSgenome.Hsapiens.UCSC.hg19)

txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
cdsByTx <- cdsBy(txdb)
chr20 <- keepSeqlevels(cdsByTx, "chr20")

## BSgenome as sequence source
bsSource <- getTranscriptSeqs(chr20, Hsapiens)
```

---

locateVariants

*Locate variants*

---

**Description**

Variant location with respect to gene function

**Usage**

```
locateVariants(query, subject, region, ...)
## S4 method for signature 'VCF,TranscriptDb,VariantType'
locateVariants(query, subject, region, ..., cache=new.env(parent=emptyenv()), ignore.strand=FALSE, asHits=)
## S4 method for signature 'GRanges,TranscriptDb,VariantType'
locateVariants(query, subject, region, ..., cache=new.env(parent=emptyenv()), ignore.strand=FALSE, asHits=)
```

**Arguments**

query	A <a href="#">Ranges</a> , <a href="#">GRanges</a> or <a href="#">VCF</a> object containing the variants. Metadata columns are allowed but are ignored.
subject	A <a href="#">TranscriptDb</a> or <a href="#">GRangesList</a> object that serves as the annotation. GFF files can be converted to <a href="#">TranscriptDb</a> objects with <a href="#">makeTranscriptDbFromGFF()</a> in the <a href="#">GenomicFeatures</a> package.
region	An instance of one of the 8 <a href="#">VariantType</a> classes : <a href="#">CodingVariants</a> , <a href="#">IntronVariants</a> , <a href="#">FiveUTRVariants</a> , <a href="#">ThreeUTRVariants</a> , <a href="#">IntergenicVariants</a> , <a href="#">SpliceSiteVariants</a> , <a href="#">PromoterVariants</a> , <a href="#">AllVariants</a> . All objects are instantiated with no arguments, e.g., <a href="#">CodingVariants()</a> will create an object of <a href="#">CodingVariants</a> . <a href="#">AllVariants</a> and <a href="#">PromoterVariants</a> classes can be given upstream and downstream arguments.
...	Additional arguments passed to methods
cache	An environment into which required components of subject are loaded. Provide, and re-use, a cache to speed repeated queries to the same subject across different query instances.
ignore.strand	A logical indicating if strand should be ignored when performing overlaps.
asHits	A logical indicating if the results should be returned as a <a href="#">Hits</a> object. Not applicable when region is <a href="#">AllVariants</a> or <a href="#">IntergenicVariants</a> .

## Details

**Range representation :** The ranges in query should reflect the position(s) of the reference allele. For snps the range will be of width 1. For range insertions or deletions the reference allele could be a sequence such as GGTG in which case the width of the range should be 4.

**Location :** Possible locations are ‘coding’, ‘intron’, ‘threeUTR’, ‘fiveUTR’, ‘intergenic’, ‘splice-Site’, or ‘promoter’.

Overlap operations for ‘coding’, ‘intron’, ‘threeUTR’, and ‘fiveUTR’ require variants to fall completely within the defined region to be classified as such.

To be classified as a ‘spliceSite’ the variant must overlap with any portion of the first 2 or last 2 nucleotides in an intron.

‘intergenic’ variants are those that do not fall within a transcript associated with a gene. If available, gene IDs for the flanking genes are give as the PRECEDEID and FOLLOWID.

‘promoter’ variants fall within a specified range upstream and downstream of the transcription start site. Ranges values can be set with the upstream and downstream arguments when creating the PromoterVariants() or AllVariants() classes.

**Subject as GRangesList :** The subject can be a TranscriptDb or GRangesList object. When using a GRangesList the type of data required is driven by the VariantType class. Below is a description of the appropriate GRangesList for each VariantType.

**CodingVariants :** coding (CDS) by transcript

**IntronVariants :** introns by transcript

**FiveUTRVariants :** five prime UTR by transcript

**ThreeUTRVariants :** three prime UTR by transcript

**IntergenicVariants :** transcripts by gene

**SpliceSiteVariants :** introns by transcript

**PromoterVariants :** list of transcripts

**AllVariants :** no GRangesList method available

**Using the cache :** When processing multiple VCF files performance is enhanced by specifying an environment as the cache argument. This cache is used to store and reuse extracted components of the subject (TxDb) required by the function. The first call to the function (i.e., processing the first VCF file in a list of many) populates the cache; repeated calls to locateVariants will access these objects from the cache vs re-extracting the same information.

## Value

A GRanges object with a row for each variant-transcript match. Columns are LOCATION, QUERYID, TXID, GENEID and CDSID.

**LOCATION** Possible locations are ‘coding’, ‘intron’, ‘threeUTR’, ‘fiveUTR’, ‘intergenic’, ‘splice-Site’ and ‘promoter’.

To be classified as ‘coding’, ‘intron’, ‘threeUTR’ or ‘fiveUTR’ the variant must fall completely within the region.

‘intergenic’ variants do not fall within a transcript. The ‘GENEID’ for these position are NA and instead the ‘PRECEDEID’ and ‘FOLLOWID’ for the flanking genes are given.

A ‘spliceSite’ variant overlaps any portion of the first 2 or last 2 nucleotides of an intron.

**QUERYID** The QUERYID column provides a map back to the row in the original query. If the query was a VCF object this index corresponds to the row in the GRanges in the rowData slot.

**TXID** The transcript id taken from the TranscriptDb object.

CDSID The coding sequence id taken from the TranscriptDb object.

GENEID The gene id taken from the TranscriptDb object.

PRECEDEID The id of the gene the query precedes. Only present for ‘intergenic’ variants.

FOLLOWID The id of the gene the query follows. Only present for ‘intergenic’ variants.

All ID values will be ‘NA’ for variants with a location of transcript\_region or NA.

### Author(s)

Valerie Obenchain <vobencha@fhcrc.org>

### See Also

[readVcf](#), [summarizeVariants](#), [predictCoding](#)

### Examples

```
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene

## -----
## TranscriptDb object as subject
## -----
## Read variants from a VCF file
fl <- system.file("extdata", "structural.vcf", package="VariantAnnotation")
vcf <- readVcf(fl, "hg19")

## Variant seqlevels do not match the TxDb seqlevels
rd <- rowData(vcf)
head(seqlevels(rd))
head(seqlevels(txdb))
intersect(seqlevels(rd), seqlevels(txdb))

## Use the renameSeqlevels helper to rename seqlevels
newnames <- paste("chr", seqlevels(vcf), sep="")
names(newnames) <- seqlevels(vcf)
vcf_adj <- renameSeqlevels(vcf, newnames)
rd_adj <- renameSeqlevels(rd, newnames)

## Confirm
intersect(seqlevels(vcf_adj), seqlevels(txdb))
intersect(seqlevels(rd_adj), seqlevels(txdb))

## VCF object as the query
loc_coding <- locateVariants(vcf_adj, txdb, AllVariants())

## -----
## GRangesList as subject
## -----
## Note the results do not include GENEID. This information
## is not available when a GRangesList is used in place of
## a TranscriptDb.
cdsbytx <- cdsBy(txdb)
locateVariants(rd_adj, cdsbytx, CodingVariants())

intbytx <- intronsByTranscript(txdb)
```

```

locateVariants(rd_adj, intbytx, IntronVariants())

## -----
## Using the cache
## -----
## Not run:
myenv <- new.env()
files <- list(vcf1, vcf2, vcf3)
lapply(files,
  function(fl) {
    vcf <- readVcf(fl, "hg19")
    ## modify seqlevels to match TxDb
    oldnms <- seqlevels(vcf)
    newnms <- paste("chr", oldnms, sep="")
    names(newnms) <- oldnms
    vcf_mod <- renameSeqlevels(vcf, newnms)
    locateVariants(vcf_mod, txdb, AllVariants(), cache=myenv)
  })

## End(Not run)

```

---

MatrixToSnpMatrix

*Convert genotype calls from a VCF file to a SnpMatrix object*


---

### Description

Convert a matrix of genotype calls from the "GT" FORMAT field of a VCF file to a [SnpMatrix](#).

### Usage

```
MatrixToSnpMatrix(callMatrix, ref, alt, ...)
```

### Arguments

callMatrix	A matrix of genotype data from the "GT" FORMAT field of a VCF file. This matrix is created with a call to readVcf and can be accessed with geno<VCF>.
ref	A DNAStrngSet of reference alleles.
alt	A DNAStrngSetList of alternate alleles.
...	Additional arguments, passed to methods.

### Details

MatrixToSnpMatrix converts a matrix of genotype calls from the "GT" FORMAT field of a VCF file into a [SnpMatrix](#). The following caveats apply,

- no distinction is made between phased and unphased genotypes
- only diploid calls are included; others are set to NA
- only single nucleotide variants are included; others are set to NA
- variants with >1 ALT allele are set to NA

In VCF files, 0 represents the reference allele and integers greater than 0 represent the alternate alleles (i.e., 2, 3, 4 would indicate the 2nd, 3rd or 4th allele in the ALT field for a particular variant).

This function only supports variants with a single alternate allele and therefore the alternate values will always be 1. Genotypes are stored in the SnpMatrix as 0, 1, 2 or 3 where 0 = missing, 1 = "0/0", 2 = "0/1" or "1/0" and 3 = "1/1". In SnpMatrix terminology, "A" is the reference allele and "B" is the risk allele. Equivalent statements to those made with 0 and 1 allele values would be 0 = missing, 1 = "A/A", 2 = "A/B" or "B/A" and 3 = "B/B".

### Value

A list with the following elements,

genotypes	The output genotype data as an object of class "SnpMatrix". The columns are snps and the rows are the samples. See the help page for <a href="#">SnpMatrix</a> for complete details of the class structure.
map	A DataFrame giving the snp names and alleles at each locus. The ignore column indicates which variants were set to NA because they met one or more of the caveats stated above.

### Author(s)

Valerie Obenchain <[vobencha@fhcrc.org](mailto:vobencha@fhcrc.org)>

### See Also

[readVcf](#), [VCF](#), [SnpMatrix](#)

### Examples

```
## Read a vcf file into a VCF object.
vcfFile <- system.file("extdata", "ex2.vcf", package="VariantAnnotation")
vcf <- readVcf(vcfFile, "hg19")

## Genotype calls are stored in the assays slot.
## Reference and alternate (variant) alleles are in ref and
## alt slots.
geno(vcf)
calls <- geno(vcf)$GT
a0 <- ref(vcf)
a1 <- alt(vcf)
mat <- MatrixToSnpMatrix(calls, a0, a1)

## The result is a list of length 2.
names(mat)

## Compare original coding in the VCF file to the SnpMatrix coding.
geno(vcf)$GT
t(as(mat$genotype, "character"))

## The 'ignore' column of the map element indicates which variants
## were set to missing as per the criteria stated in the details section.
mat$map

## Variant rs6040355 was ignored because it has more than one alternate
## allele. The microsat1 variant has a reference allele with length
## greater than 1 and more than 1 alternate allele.
## Variant chr20:1230237 was ignored because the alternate allele is not
## of length 1.
```

```
## View the reference and alternate alleles.
DataFrame(reference = ref(vcf),
           alternate = CharacterList(as.list(alt(vcf))),
           row.names = rownames(vcf))
```

---

PolyPhenDb-class

*PolyPhenDb objects*


---

## Description

The PolyPhenDb class is a container for storing a connection to a PolyPhen sqlite database.

## Details

PolyPhen (Polymorphism Phenotyping) is a tool which predicts the possible impact of an amino acid substitution on the structure and function of a human protein by applying empirical rules to the sequence, phylogenetic and structural information characterizing the substitution.

PolyPhen makes its predictions using UniProt features, PSIC profiles scores derived from multiple alignment and matches to PDP or PQS structural databases. The procedure can be roughly outlined in the following steps, see the references for complete details,

- sequence-based characterization of substitution site
- calculation of PSIC profile scores for two amino acid variants
- calculation of structural parameters and contacts
- prediction

PolyPhen uses empirically derived rules to predict that a non-synonymous SNP is

- probably damaging : it is with high confidence supposed to affect protein function or structure
- possibly damaging : it is supposed to affect protein function or structure
- benign : most likely lacking any phenotypic effect
- unknown : when in some rare cases, the lack of data do not allow PolyPhen to make a prediction

## Methods

In the code below, x is a PolyPhenDb object.

`metadata(x)`: Returns x's metadata in a data frame.

`cols(x)`: Returns the names of the cols that can be used to subset the data columns. For column descriptions see `?PolyPhenDbColumns`.

`keys(x)`: Returns the names of the keys that can be used to subset the data rows. The keys values are the rsid's.

`select(x, keys = NULL, cols = NULL, ...)`: Returns a subset of data defined by the character vectors keys and cols. If no keys are supplied, all rows are returned. If no cols are supplied, all columns are returned. See `?PolyPhenDbColumns` for column descriptions.

`duplicateRSID(x)`: Returns a named list of duplicate rsid groups. The names are the keys, the list elements are the rsid's that have been reported as having identical chromosome position and alleles and therefore translating into the same amino acid residue substitution.



**Author(s)**

Valerie Obenchain <vobencha@fhcrc.org>

**References**

PolyPhen Home: <http://genetics.bwh.harvard.edu/pph2/dokuwiki/>

Adzhubei IA, Schmidt S, Peshkin L, Ramensky VE, Gerasimova A, Bork P, Kondrashov AS, Sunyaev SR. Nat Methods 7(4):248-249 (2010).

Ramensky V, Bork P, Sunyaev S. Human non-synonymous SNPs: server and survey. Nucleic Acids Res 30(17):3894-3900 (2002).

Sunyaev SR, Eisenhaber F, Rodchenkov IV, Eisenhaber B, Tumanyan VG, Kuznetsov EN. PSIC: profile extraction from sequence alignments with position-specific counts of independent observations. Protein Eng 12(5):387-394 (1999).

**See Also**

?PolyPhenDbColumns

**Examples**

```
library(PolyPhen.Hsapiens.dbSNP131)

## metadata
metadata(PolyPhen.Hsapiens.dbSNP131)

## available rsid's
head(keys(PolyPhen.Hsapiens.dbSNP131))

## column descriptions found at ?PolyPhenDbColumns
cols(PolyPhen.Hsapiens.dbSNP131)

## subset on keys and cols
subst <- c("AA1", "AA2", "PREDICTION")
rsids <- c("rs2142947", "rs4995127", "rs3026284")
select(PolyPhen.Hsapiens.dbSNP131, keys=rsids, cols=subst)

## retrieve substitution scores
subst <- c("IDPMAX", "IDPSNP", "IDQMIN")
select(PolyPhen.Hsapiens.dbSNP131, keys=rsids, cols=subst)

## retrieve the PolyPhen-2 classifiers
subst <- c("PPH2CLASS", "PPH2PROB", "PPH2FPR", "PPH2TPR", "PPH2FDR")
select(PolyPhen.Hsapiens.dbSNP131, keys=rsids, cols=subst)

## duplicate groups of rsid's
duplicateRSID(PolyPhen.Hsapiens.dbSNP131, c("rs71225486", "rs1063796"))
```

---

PolyPhenDbColumns     *PolyPhenDb Columns*

---

## Description

Description of the PolyPhen Sqlite Database Columns

### Column descriptions

These column names are displayed when cols is called on a PolyPhenDb object.

- rsid : rsid

Original query :

- OSNPID : original SNP identifier from user input
- OSNPACC : original protein identifier from user input
- OPOS : original substitution position in the protein sequence from user input
- OAA1 : original wild type (reference) aa residue from user input
- OAA2 : original mutant (reference) aa residue from user input

Mapped query :

- SNPID : SNP identifier mapped to dbSNP rsID if available, otherwise same as o\_snp\_id. This value was used as the rsid column
- ACC : protein UniProtKB accession if known protein, otherwise same as o\_acc
- POS : substitution position mapped to UniProtKB protein sequence if known, otherwise same as o\_pos
- AA1 : wild type aa residue
- AA2 : mutant aa residue
- NT1 : wild type allele nucleotide
- NT2 : mutant allele nucleotide

PolyPhen-2 prediction :

- PREDICTION : qualitative ternary classification FPR thresholds

PolyPhen-1 prediction :

- BASEDON : prediction basis
- EFFECT : predicted substitution effect on the protein structure or function

PolyPhen-2 classifiers :

- PPH2CLASS : binary classifier outcome ("damaging" or "neutral")
- PPH2PROB : probability of the variation being dammaging
- PPH2FPR : false positive rate at the pph2\_prob level
- PPH2TPR : true positive rate at the pph2\_prob level
- PPH2FDR : false discovery rate at the pph2\_prob level

UniProtKB-SwissProt derived protein sequence annotations :

- SITE : substitution SITE annotation
- REGION : substitution REGION annotation
- PHAT : PHAT matrix element for substitution in the TRANSMEM region

Multiple sequence alignment scores :

- DSCORE : difference of PSIC scores for two aa variants (Score1 - Score2)
- SCORE1 : PSIC score for wild type aa residue (aa1)
- SCORE2 : PSIC score for mutant aa residue (aa2)
- NOBS : number of residues observed at the substitution position in the multiple alignment (sans gaps)

Protein 3D structure features :

- NSTRUCT : initial number of BLAST hits to similar proteins with 3D structures in PDB
- NFILT : number of 3D BLAST hits after identity threshold filtering
- PDBID : protein structure identifier from PDB
- PDBPOS : position of substitution in PDB protein sequence
- PDBCH : PDB polypeptide chain identifier
- IDENT : sequence identity between query and aligned PDB sequences
- LENGTH : PDB sequence alignment length
- NORMACC : normalized accessible surface
- SECSTR : DSSP secondary structure assignment
- MAPREG : region of the phi-psi (Ramachandran) map derived from the residue dihedral angles
- DVOL : change in residue side chain volume
- DPROP : change in solvent accessible surface propensity resulting from the substitution
- BFACT : normalized B-factor (temperature factor) for the residue
- HBONDS : number of hydrogen sidechain-sidechain and sidechain-mainchain bonds formed by the residue
- AVENHET : average number of contacts with heteroatoms per residue
- MINDHET : closest contact with heteroatom
- AVENINT : average number of contacts with other chains per residue
- MINDINT : closest contact with other chain
- AVENSIT : average number of contacts with critical sites per residue
- MINDSIT : closest contact with a critical site

Nucleotide sequence features (CpG/codon/exon junction) :

- TRANSV : whether substitution is a transversion
- CODPOS : position of the substitution within the codon
- CPG : whether or not the substitution changes CpG context
- MINDJNC : substitution distance from exon/intron junction

Pfam protein family :

- PFAMHIT : Pfam identifier of the query protein

Substitution scores :

- IDPMAX : maximum congruency of the mutant aa residue to all sequences in multiple alignment
- IDPSNP : maximum congruency of the mutant aa residue to the sequence in alignment with the mutant residue
- IDQMIN : query sequence identity with the closest homologue deviating from the wild type aa residue

Comments :

- COMMENTS : Optional user comments

### Author(s)

Valerie Obenchain <vobencha@fhcrc.org>

### See Also

?PolyPhenDb

---

predictCoding

*Predict amino acid coding changes for variants*

---

### Description

Predict amino acid coding changes for variants a coding regions

### Usage

```
## S4 method for signature 'VCF,TranscriptDb,ANY,missing'
predictCoding(query, subject, seqSource, varAllele, ..., ignore.strand=FALSE)
## S4 method for signature 'Ranges,TranscriptDb,ANY,DNAStringSet'
predictCoding(query, subject, seqSource, varAllele, ..., ignore.strand=FALSE)
## S4 method for signature 'GRanges,TranscriptDb,ANY,DNAStringSet'
predictCoding(query, subject, seqSource, varAllele, ..., ignore.strand=FALSE)
```

### Arguments

query	A <a href="#">VCF</a> , <a href="#">Ranges</a> or <a href="#">GRanges</a> instance containing the variants to be annotated. If a <a href="#">Ranges</a> is provided it will be coerced to a <a href="#">GRanges</a> . If a <a href="#">VCF</a> is provided the <a href="#">GRanges</a> from the rowData slot will be used. All elementMetadata columns are ignored.  When the query is not a VCF object a varAllele must be provided. The varAllele must be a DNAStringSet the same length as the query. If there are multiple alternate alleles per variant the query must be expanded to one row per each alternate allele. See examples.
subject	A <a href="#">TranscriptDb</a> object that serves as the annotation. GFF files can be converted to <a href="#">TranscriptDb</a> objects with makeTranscriptDbFromGFF() in the GenomicFeatures package.

seqSource	A <a href="#">BSgenome</a> instance or an <a href="#">FaFile</a> to be used for sequence extraction.
varAllele	A <a href="#">DNAStringSet</a> containing the variant (alternate) alleles. The length of varAllele must equal the length of query. Missing values are represented by a zero width empty element. Ranges with missing varAllele values are ignored; those with an 'N' character are not translated. When the query is a VCF object the varAllele argument will be missing. This value is taken internally from the VCF with alt(<VCF>).
...	Additional arguments passed to methods.
ignore.strand	A logical indicating if strand should be ignored when performing overlaps.

### Details

This function returns the amino acid coding for variants that fall in coding regions. The reference sequences are taken from a fasta file or [BSgenome](#). The width of the reference is determined from the start position and width of the range in the query. For guidance on how to represent an insertion, deletion or substitution see the 1000 Genomes VCF format (references).

Variant alleles are taken from the varAllele when supplied. When the query is a VCF object the varAllele will be missing. This value is taken internally from the VCF with alt(<VCF>). The variant allele is substituted into the reference sequences and transcribed. Transcription only occurs if the substitution, insertion or deletion results in a new sequence length divisible by 3.

When the query is an unstranded (\*) GRanges predictCoding will attempt to find overlaps on both the positive and negative strands of the subject. When the subject hit is on the negative strand the return varAllele is reverse complemented. The strand of the returned GRanges represents the strand of the subject hit.

### Value

A [GRanges](#) with a row for each variant-transcript match. The result includes only variants that fall within coding regions. If the query was unstranded (\*) or ignore.strand=TRUE the strand of the output GRanges represents the strand of the subject hit.

At a minimum, the elementMetadata columns include,

varAllele Variant allele. This value is reverse complemented for an unstranded query that overlaps a subject on the negative strand.

QUERYID Map back to the row in the original query

TXID Internal transcript id from the annotation

CDSID Internal coding region id from the annotation

GENEID Internal gene id from the annotation

CDSLOC Location in coding region-based coordinates of the first nucleotide in the variant.

PROTEINLOC Location in cds-based coordinates of the first nucleotide in the variant. This position is relative to the start of the cds region defined in the subject annotation.

CONSEQUENCE Possible values are 'synonymous', 'nonsynonymous', 'frameshift', 'nonsense' and 'not translated'. Variant sequences are translated only when the codon sequence is a multiple of 3. The value will be 'frameshift' when a sequence is of incompatible length and it will be 'not translated' when the varAllele is missing or there is an 'N' in the sequence. 'nonsense' is used for premature stop codons.

REFCODON The reference codon sequence. This range is typically greater than the width of the range in the GRanges because it includes all codons involved in the sequence modification. If the reference sequence is of width 2 but the alternate allele is of width 4 then at least two codons must be included in the REFSEQ.

**VARCODON** This sequence is the result of inserting, deleting or replacing the position(s) in the reference sequence alternate allele. If the result of this modification is not a multiple of 3 no translation is performed and the VARAA value will be missing.

**REEFAA** The reference amino acid column contains the translated REFSEQ.

**VARAA** The variant amino acid column contains the translated VARSEQ. When translation is not possible this value is missing.

### Author(s)

Michael Lawrence and Valerie Obenchain <[vobencha@fhcrc.org](mailto:vobencha@fhcrc.org)>

### References

<http://vcftools.sourceforge.net/specs.html>

### See Also

[readVcf](#), [locateVariants](#), [refLocsToLocalLocs](#) [getTranscriptSeqs](#)

### Examples

```
library(BSgenome.Hsapiens.UCSC.hg19)
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene

## -----
## VCF object as query
## -----
## Read variants from a VCF file
fl <- system.file("extdata", "chr22.vcf.gz", package="VariantAnnotation")
vcf <- readVcf(fl, "hg19")

## Rename seqlevels in the VCF object to match those in the TxDb
vcf<- renameSeqlevels(vcf, c("22"="chr22"))
## Confirm common seqlevels
intersect(seqlevels(vcf), seqlevels(txdb))

## Note when 'query' is a VCF object the varAllele argument is missing.
coding1 <- predictCoding(vcf, txdb, Hsapiens)
head(coding1, 3)

## -----
## GRanges object as query
## -----
## Alternatively, a GRanges can be the 'query' to predictCoding().
## The seqlevels were previously adjusted in the VCF object so the GRanges
## extracted from rowData() has the correct seqlevels.
rd <- rowData(vcf)

## The GRanges must be expanded to have one row per alternate allele.
## Variants 1, 2 and 10 have two alternate alleles.
altallele <- alt(vcf)
eltlen <- elementLengths(altallele)
rd_exp <- rep(rd, eltlen)

## Call predictCoding() with the expanded GRanges as the 'query'
```

```
## and the unlisted alternate allele as the 'varAllele'.
coding2 <- predictCoding(rd_exp, txdb, Hsapiens, unlist(altallele))

identical(coding1, coding2)
```

---

readVcf

*Read VCF files*


---

## Description

Read Variant Call Format (VCF) files

## Usage

```
## S4 method for signature 'TabixFile,character,ScanVcfParam'
readVcf(file, genome, param, ...)
## S4 method for signature 'TabixFile,character,RangedData'
readVcf(file, genome, param, ...)
## S4 method for signature 'TabixFile,character,RangesList'
readVcf(file, genome, param, ...)
## S4 method for signature 'TabixFile,character,GRanges'
readVcf(file, genome, param, ...)
## S4 method for signature 'TabixFile,character,missing'
readVcf(file, genome, param, ...)
## S4 method for signature 'character,character,ScanVcfParam'
readVcf(file, genome, param, ...)
## S4 method for signature 'character,character,missing'
readVcf(file, genome, param, ...)
## S4 method for signature 'character,missing,missing'
readVcf(file, genome, param, ...)
```

## Arguments

- |        |   |
|--------|---|
| file   | A <a href="#">TabixFile</a> instance or character() name of the VCF file to be processed. When ranges are specified in param, file must be a <a href="#">TabixFile</a> .<br>Use of the <a href="#">TabixFile</a> methods are encouraged as they are more efficient than the character() methods. See ?TabixFile and ?indexTabix for help creating a <a href="#">TabixFile</a> . |
| genome | The character() name of the genome the variants are mapped to. This information is stored in the genome slot of the seqinfo associated with the <a href="#">GRanges</a> object in rowData.  |
| param  | A instance of <a href="#">ScanVcfParam</a> , <a href="#">GRanges</a> , <a href="#">RangedData</a> or <a href="#">RangesList</a> . VCF files can be subset on genomic coordinates (ranges) or elements in the VCF fields. Both genomic coordinates and VCF elements can be specified in a <a href="#">ScanVcfParam</a> . See ?ScanVcfParam for details.                          |
| ...    | Additional arguments, passed to methods.  |

## Details

**Data Import : VCF object :** readVcf imports records from bzip compressed or uncompressed VCF files. Data are parsed into a **VCF** object using the file header information if available. To import a subset of ranges the VCF must have a Tabix index file. An index file can be created with bzip and indexTabix functions. See examples.

readVcf calls **scanVcf**, the details of which can be found with ?scanVcf.

**Data type :** CHROM, POS, ID and REF fields are used to create the GRanges stored in the rowData slot of the VCF object. Access with rowData accessor.

REF, ALT, QUAL and FILTER are parsed into the DataFrame in the fixed slot. Because ALT can have more than one value per variant it is represented as a DNAStrngSetList. REF is a DNAStrngSet, QUAL is numeric and FILTER is a character. Accessors include fixed, ref, alt, qual, and filt.

Data from the INFO field can be accessed with the info accessor. Genotype data (i.e., data immediately following the FORMAT field in the VCF) can be accessed with the geno accessor. INFO and genotype data types are determined according to the 'Number' and 'Type' information in the file header as follows :

If 'Number' is 1, 'info' data are parsed into a vector. 'geno' data are parsed into a matrix where the columns are the samples.

If 'Number' is an integer >1, 'info' data are parsed into a DataFrame with the indicated number of columns. 'geno' are parsed into an array with the same dimensions as 'Number'. Columns of the 'geno' matrices are the samples.

If 'Number' is '.', 'A' or 'G', a matrix is used for both 'info' and 'geno' data.

When the VCF header does not contain data type information, the data are returned as a single unparsed column named 'INFO' or 'GENO'.

**Missing data :** Missing data in VCF files are represented by a dot ("."). readVcf retains the dot as a character string for data type character and converts it to NA for data types numeric or double.

Because the data are stored in rectangular data structures there is a value for each info and geno field element in the VCF class. If the element was missing or was not collected for a particular variant the value will be NA.

## Value

The object returned is a **VCF** class instance. See ?VCF for complete details of the class structure.

**rowData:** The CHROM, POS, ID and REF fields are used to create a GRanges object. The ranges are created using POS as the start value and width of the reference allele (REF). The IDs become the rownames. If they are missing (i.e., '.') a colon separated string of chromosome and start position will be used instead. The genome argument is stored in the seqinfo of the GRanges and can be accessed with genome(<VCF>).

One elementMetadata column, paramRangeID, is included with the rowData. This ID is meaningful when multiple ranges are specified in the ScanVcfParam. This ID distinguishes which records match each range.

**fixed:** REF, ALT, QUAL and FILTER fields of the VCF are parsed into a DataFrame.

**info:** Data from the INFO field of the VCF is parsed into a DataFrame.

**geno:** If present, the genotype data are parsed into a list of matrices or arrays. Each list element represents a field in the FORMAT column of the VCF file. Rows are the variants, columns are the samples.

**colData:** This slot contains a DataFrame describing the samples. If present, the sample names following FORMAT in the VCF file become the row names.



**exptData:** Header information present in the file is put into a SimpleList in exptData.

See references for complete details of the VCF file format.

### Author(s)

Valerie Obenchain <vobencha@fhcrc.org>

### References

<http://vcftools.sourceforge.net/specs.html> outlines the VCF specification.

<http://samtools.sourceforge.net/mpileup.shtml> contains information on the portion of the specification implemented by bcftools.

<http://samtools.sourceforge.net/> provides information on samtools.

### See Also

[indexTabix](#), [TabixFile](#), [scanTabix](#), [scanBcf](#), [readVcfLongForm](#)

### Examples

```
fl <- system.file("extdata", "ex2.vcf", package="VariantAnnotation")
vcf <- readVcf(fl, "hg19")

## -----
## Header and genome information
## -----
vcf

## extract the VCFHeader object from exptData
hdr <- exptData(vcf)[["header"]]
## use accessors to extract the data
info(hdr)
fixed(hdr)

## genome information is stored in the GRanges
unique(genome(rowData(vcf)))

## -----
## Accessors
## -----
## fixed fields together
head(fixed(vcf), 5)

## fixed fields separately
filt(vcf)
ref(vcf)

## info data
info(hdr)
info(vcf)
values(info(vcf))[["DP"]]

## geno data
geno(hdr)
geno(vcf)
```

```

head(geno(vcf)$GT)

identical(geno(vcf)[["GQ"]], geno(vcf)$GQ)

## -----
## Data subsets
## -----

## Subset on genome coordinates :
## 'file' must have a Tabix index
rngs <- GRanges("20", IRanges(c(14370, 1110000), c(17330, 1234600)))
names(rngs) <- c("geneA", "geneB")
param <- ScanVcfParam(which=rngs)
compressVcf <- bgzip(fl, tempfile())
idx <- indexTabix(compressVcf, "vcf")
tab <- TabixFile(compressVcf, idx)
vcf <- readVcf(tab, "hg19", param)

## 'paramRangeID' associates the records to the ranges in the param
rowData(vcf)

## Subset on 'fixed', 'info' or 'geno' VCF elements :
param <- ScanVcfParam(fixed="ALT", geno=c("GT", "HQ"), info=c("NS", "AF"))
vcf_tab <- readVcf(tab, "hg19", param)
vcf_fname <- readVcf(fl, "hg19", param)
identical(vcf_tab, vcf_fname)

## Subset on both genome coordinates and VCF elements :
param <- ScanVcfParam(geno="HQ", info="AF", which=rngs)
vcf <- readVcf(tab, "hg19", param)

## When any of 'fixed', 'info' or 'geno' are omitted (i.e., no
## elements specified) all records are retrieved. Use NA to indicate
## that no records should be retrieved. This param specifies
## all 'fixed' fields, the "GT" 'geno' field and none of 'info'.
ScanVcfParam(geno="GT", info=NA)

## iterate through VCF file
fl <- system.file("extdata", "chr22.vcf.gz", package="VariantAnnotation")
param <- ScanVcfParam(fixed="ALT", geno=c("GT", "GL"), info=c("LDAF", "AF"))
tab <- TabixFile(fl, yieldSize=4000)
open(tab)
while (nrow(vcf <- readVcf(tab, "hg19", param=param)))
  cat("vcf dim:", dim(vcf), "\n")
close(tab)

```

---

readVcfLongForm

*Read VCF files into a long form GRanges*


---

## Description

Read Variant Call Format (VCF) files into a long form GRanges

**Usage**

```

## S4 method for signature 'TabixFile,character,ScanVcfParam'
readVcfLongForm(file, genome, param, ...)
## S4 method for signature 'TabixFile,character,RangedData'
readVcfLongForm(file, genome, param, ...)
## S4 method for signature 'TabixFile,character,RangesList'
readVcfLongForm(file, genome, param, ...)
## S4 method for signature 'TabixFile,character,GRanges'
readVcfLongForm(file, genome, param, ...)
## S4 method for signature 'TabixFile,character,missing'
readVcfLongForm(file, genome, param, ...)
## S4 method for signature 'character,character,ScanVcfParam'
readVcfLongForm(file, genome, param, ...)
## S4 method for signature 'character,character,missing'
readVcfLongForm(file, genome, param, ...)
## S4 method for signature 'character,missing,missing'
readVcfLongForm(file, genome, param, ...)

```

**Arguments**

file	A <a href="#">TabixFile</a> instance or character() name of the VCF file to be processed. When ranges are specified in param, file must be a <a href="#">TabixFile</a> . Use of the <a href="#">TabixFile</a> methods are encouraged as they are more efficient than the character() methods. See ?TabixFile and ?indexTabix for help creating a <a href="#">TabixFile</a> .
genome	The character() name of the genome the variants are mapped to. This information is stored in the genome slot of the seqinfo associated with the <a href="#">GRanges</a> object in rowData.
param	A instance of <a href="#">ScanVcfParam</a> , <a href="#">GRanges</a> , <a href="#">RangedData</a> or <a href="#">RangesList</a> . VCF files can be subset on genomic coordinates (ranges) or elements in the VCF fields. Both genomic coordinates and VCF elements can be specified in a <a href="#">ScanVcfParam</a> . See ?ScanVcfParam for details.
...	Additional arguments, passed to methods.

**Details**

**Long Form GRanges object :** readVcfLongForm reads data from a VCF file in the same manner as readVcf. Input arguments and the ability to subset the data on ranges or VCF fields are the same. The return object is a long form GRanges expanded to the length of the ‘unlisted’ alternate allele (ALT). The fixed and info data are also expanded as elementMetadata columns. Currently no geno information is included.

**Value**

A GRanges of variant locations. When the alternate allele column (ALT) column in the VCF file has a single value per variant the GRanges will have a single row per variant (i.e., not expanded). When variants have multiple alternate allele values the GRanges will have repeated rows for the variants with multiple values.

**Author(s)**

Valerie Obenchain <vobencha@fhcrc.org>

**References**

<http://vcftools.sourceforge.net/specs.html> outlines the VCF specification.

<http://samtools.sourceforge.net/mpileup.shtml> contains information on the portion of the specification implemented by bcftools.

<http://samtools.sourceforge.net/> provides information on samtools.

**See Also**

[readVcf](#)

**Examples**

```
## All data
fl <- system.file("extdata", "ex2.vcf", package="VariantAnnotation")
gr1 <- readVcfLongForm(fl, "hg19")

## Subset on ranges :
rngs <- GRanges("20", IRanges(c(14370, 1234567), c(17330, 1234567)))
names(rngs) <- c("geneA", "geneB")
param <- ScanVcfParam(which=rngs)
compressVcf <- bgzip(fl, tempfile())
idx <- indexTabix(compressVcf, "vcf")
tab <- TabixFile(compressVcf, idx)
gr2 <- readVcfLongForm(tab, "hg19", param)

## 'paramRangeID' associates the records to the ranges in the param
gr2

## Subset on 'fixed' or 'info' VCF elements :
param <- ScanVcfParam(which=rngs, fixed="ALT", info=c("NS", "AF"))
gr3 <- readVcfLongForm(tab, "hg19", param)
```

---

refLocsToLocalLocs      *refLocsToLocalLocs*

---

**Description**

Converts reference locations (aka chromosome-based or genomic) to coding regions, and protein based locations

**Usage**

```
refLocsToLocalLocs(ranges, txdb, cdsbytx, ...)
## S4 method for signature 'GRanges,TranscriptDb,missing'
refLocsToLocalLocs(ranges, txdb, cdsbytx, ...)
## S4 method for signature 'GRanges,missing,GRangesList'
refLocsToLocalLocs(ranges, txdb, cdsbytx, ...)
```

**Arguments**

ranges	A <a href="#">GRanges</a> object containing the variants in reference-based coordinates.
txdb	A <a href="#">TranscriptDb</a> object that serves as the annotation reference.
cdsbytx	A <a href="#">GRangesList</a> object with transcripts as the outer list elements and coding regions as the inner.
...	Additional arguments passed to methods

**Details**

This function translates the reference-based coordinates in ranges to ‘local’ coordinates in the coding region (CDS) and protein sequences.

When a txdb is supplied the cdsbytx is created with cdsBy(). If cdsbytx is provided the outer list elements must be transcripts and the inner list elements represent coding regions. The GRangesList objects must have names on the outer list elements (i.e., transcript names).

Only ranges that fall ‘within’ coding sequences are reported in the result. Output is a modified GRanges of the ranges input where each row represents a range-transcript match making multiple rows per range possible. The elementMetadata columns include tx\_id, cdsLoc and proteinLoc. When a txdb is provided the txId is the internal transcript id from the annotation. When cdsbytx is provided tx\_id are the names on the outer list elements.

If ranges is unstranded the strand of the return value reflects the strand of the subject the ranges overlapped with.

**Value**

A [GRanges](#) with the following elementMetadata columns,

CDSLOC Location in coding region coordinates

PROTEINLOC Location in protein (codon triplet) coordinates

QUERYID Character vector mapping to the of rows of the original query

TXID Character vector of internal transcript ids from the [TranscriptDb](#) or the names of the outer list elements of the cdsbytx object.

CDSID Character vector of internal coding region ids from the [TranscriptDb](#) or the names of the outer list elements of the cdsbytx object.

**Author(s)**

Michael Lawrence and Valerie Obenchain <[vobencha@fhcrc.org](mailto:vobencha@fhcrc.org)>

**See Also**

[map](#) [predictCoding](#) [getTranscriptSeqs](#) [transcriptLocs2refLocs](#) [extractTranscriptsFromGenome](#)

**Examples**

```
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
ranges <- GRanges(c("chr12", "chr1", "chr5"),
                  IRanges(c(1017956, 881906, 140532),
                          c(1017956, 881907, 140532)))
refLocsToLocalLocs(ranges, TxDb.Hsapiens.UCSC.hg19.knownGene)
```

---

 scanVcf

*Import VCF files*


---

## Description

Import Variant Call Format (VCF) files in text or binary format

## Usage

```

scanVcfHeader(file, ...)
## S4 method for signature 'character'
scanVcfHeader(file, ...)

scanVcf(file, ..., param)
## S4 method for signature 'character,ScanVcfParam'
scanVcf(file, ..., param)
## S4 method for signature 'character,missing'
scanVcf(file, ..., param)
## S4 method for signature 'connection,missing'
scanVcf(file, ..., param)

## S4 method for signature 'TabixFile'
scanVcfHeader(file, ...)
## S4 method for signature 'TabixFile,missing'
scanVcf(file, ..., param)
## S4 method for signature 'TabixFile,ScanVcfParam'
scanVcf(file, ..., param)
## S4 method for signature 'TabixFile,GRanges'
scanVcf(file, ..., param)
## S4 method for signature 'TabixFile,RangedData'
scanVcf(file, ..., param)
## S4 method for signature 'TabixFile,RangesList'
scanVcf(file, ..., param)

## S4 method for signature 'TabixFile'
scanVcfHeader(file, ...)
## S4 method for signature 'TabixFile,missing'
scanVcf(file, ..., param)
## S4 method for signature 'TabixFile,ScanVcfParam'
scanVcf(file, ..., param)
## S4 method for signature 'TabixFile,GRanges'
scanVcf(file, ..., param)
## S4 method for signature 'TabixFile,RangedData'
scanVcf(file, ..., param)
## S4 method for signature 'TabixFile,RangesList'
scanVcf(file, ..., param)

```

## Arguments

file                    For scanVcf and scanVcfHeader, the character() file name, [TabixFile](#), or class connection (file() or bgzip()) of the 'VCF' file to be processed.

param	A instance of <a href="#">ScanVcfParam</a> influencing which records are parsed and the 'INFO' and 'GENO' information returned.
...	Additional arguments for methods

### Details

The argument param allows portions of the file to be input, but requires that the file be bgzip'd and indexed as a [TabixFile](#).

scanVcf with param="missing" and file="character" or file="connection" scan the entire file. With file="connection", an argument n indicates the number of lines of the VCF file to input; a connection open at the beginning of the call is open and incremented by n lines at the end of the call, providing a convenient way to stream through large VCF files.

The INFO field of the scanned VCF file is returned as a single 'packed' vector, as in the VCF file. The GENO field is a list of matrices, each matrix corresponds to a field as defined in the FORMAT field of the VCF header. Each matrix has as many rows as scanned in the VCF file, and as many columns as there are samples. As with the INFO field, the elements of the matrix are 'packed'. The reason that INFO and GENO are returned packed is to facilitate manipulation, e.g., selecting particular rows or samples in a consistent manner across elements.

### Value

scanVcfHeader returns a list, with one element for each file named in file. Each element of the list is itself a list containing three elements. The reference element is a character() vector with names of reference sequences. The sample element is a character() vector of names of samples. The header element is a character() vector of the header lines (preceeded by "##") present in the VCF file.

scanVcf returns a list, with one element per range. Each list has 7 elements, obtained from the columns of the VCF specification:

**rowData** GRanges instance derived from CHROM, POS, ID, and the width of REF

**REF** reference allele

**ALT** alternate allele

**QUAL** phred-scaled quality score for the assertion made in ALT

**FILTER** indicator of whether or not the position passed all filters applied

**INFO** additional information

**GENO** genotype information immediately following the FORMAT field in the VCF

The GENO element is itself a list, with elements corresponding to those defined in the VCF file header. For scanVcf, elements of GENO are returned as a matrix of records x samples; if the description of the element in the file header indicated multiplicity other than 1 (e.g., variable number for "A", "G", or "."), then each entry in the matrix is a character string with sub-entries comma-delimited.

### Author(s)

Martin Morgan and Valerie Obenchain <[vobencha@fhcrc.org](mailto:vobencha@fhcrc.org)>

### References

<http://vcftools.sourceforge.net/specs.html> outlines the VCF specification.

<http://samtools.sourceforge.net/mpileup.shtml> contains information on the portion of the specification implemented by bcftools.

<http://samtools.sourceforge.net/> provides information on samtools.

**See Also**

[readVcf](#) [BcfFile](#) [TabixFile](#)

**Examples**

```
fl <- system.file("extdata", "ex2.vcf", package="VariantAnnotation")
scanVcfHeader(fl)
vcf <- scanVcf(fl)
## value: list-of-lists
str(vcf)
names(vcf[[1]][["GENO"]])
vcf[[1]][["GENO"]][["GT"]]
```

---

ScanVcfParam-class      *Parameters for scanning VCF files*

---

**Description**

Use `ScanVcfParam()` to create a parameter object influencing which records and fields are imported from a VCF file. Record parsing is based on genomic coordinates and requires a Tabix index file. Individual VCF elements can be specified in the ‘fixed’, ‘info’ and ‘geno’ arguments.

**Usage**

```
ScanVcfParam(fixed=character(), info=character(), geno=character(),
             trimEmpty=TRUE, which, ...)
```

```
## Getters and Setters
```

```
vcfFixed(object)
vcfFixed(object) <- value
vcfInfo(object)
vcfInfo(object) <- value
vcfGeno(object)
vcfGeno(object) <- value
vcfTrimEmpty(object)
vcfTrimEmpty(object) <- value
vcfWhich(object)
vcfWhich(object) <- value
```

**Arguments**

fixed	A <code>character()</code> vector of fixed fields to be returned. Possible values are ALT, QUAL and FILTER. The CHROM, POS, ID and REF fields are needed to create the GRanges of variant locations. Because these are essential fields there is no option to request or omit them. If not specified, all fields are returned; if <code>fixed=NA</code> only REF is returned.
info	A <code>character()</code> vector naming the ‘INFO’ fields to return. <code>scanVcfHeader()</code> returns a vector of available fields. If not specified, all fields are returned; if <code>info=NA</code> no fields are returned.



geno	A character() vector naming the 'GENO' fields to return. scanVcfHeader() returns a vector of available fields. If not specified, all fields are returned; if geno=NA no fields are returned.
trimEmpty	A logical(1) indicating whether 'GENO' fields with no values should be returned.
which	A GRanges, RangesList, RangedData or missing object, describing the sequences and ranges to be queried. Variants whose POS lies in the interval(s) [start, end] are returned. If which is not specified all ranges are returned.
object	An instance of class ScanVcfParam.
value	An instance of the corresponding slot, to be assigned to object.
...	Arguments passed to methods.

### Objects from the Class

Objects can be created by calls of the form ScanVcfParam().

### Slots

**which:** Object of class "RangesList" indicating which reference sequence and coordinate variants must overlap.

**fixed:** Object of class "character" indicating fixed fields to be returned.

**info:** Object of class "character" indicating portions of 'INFO' to be returned.

**geno:** Object of class "character" indicating portions of 'GENO' to be returned.

**trimEmpty:** Object of class "logical" indicating whether empty 'GENO' fields are to be returned.

### Functions and methods

See 'Usage' for details on invocation.

Constructor:

**ScanVcfParam:** Returns a ScanVcfParam object. The which argument to the constructor can be one of several types, as documented above.

Accessors:

**vcfFixed, vcfInfo, vcfGeno, vcfTrimEmpty, vcfWhich:** Return the corresponding field from object.

Methods:

**show** Compactly display the object.

### Author(s)

Martin Morgan and Valerie Obenchain <vobencha@fhcrc.org>

### See Also

[readVcf](#)

**Examples**

```

ScanVcfParam()

## -----
## 'which' argument
## -----
## To subset on genomic coordinates, create a GRanges, RangedData or
## RangesList with the ranges of interest. This object is supplied
## to ScanVcfParam() as the 'which' argument.
which <- RangesList(seq1=IRanges(1000, 2000),
  seq2=IRanges(c(100, 1000), c(1000, 2000)))
ScanVcfParam(which=which)

## -----
## 'fixed', 'info' and 'geno' arguments
## -----
## This param specifies the return of the "GT" 'geno' field and the
## subset of ranges in 'which'. All 'fixed' and 'info' fields will be
## returned.
ScanVcfParam(geno="GT", which=which)

## Here two 'fixed' and one 'geno' field are specified
ScanVcfParam(fixed=c("ALT", "QUAL"), geno="GT", info=NA)

## Return only the 'fixed' fields
ScanVcfParam(geno=NA, info=NA)

```

---

SIFTDb-class

*SIFTDb objects*


---

**Description**

The SIFTDb class is a container for storing a connection to a SIFT sqlite database.

**Details**

SIFT is a sequence homology-based tool that sorts intolerant from tolerant amino acid substitutions and predicts whether an amino acid substitution in a protein will have a phenotypic effect. SIFT is based on the premise that protein evolution is correlated with protein function. Positions important for function should be conserved in an alignment of the protein family, whereas unimportant positions should appear diverse in an alignment.

SIFT uses multiple alignment information to predict tolerated and deleterious substitutions for every position of the query sequence. The procedure can be outlined in the following steps,

- search for similar sequences
- choose closely related sequences that may share similar function to the query sequence
- obtain the alignment of the chosen sequences
- calculate normalized probabilities for all possible substitutions from the alignment.

Positions with normalized probabilities less than 0.05 are predicted to be deleterious, those greater than or equal to 0.05 are predicted to be tolerated.

## Methods

In the code below, `x` is a SIFTDb object.

`metadata(x)`: Returns `x`'s metadata in a data frame.

`cols(x)`: Returns the names of the cols that can be used to subset the data columns.

`keys(x)`: Returns the names of the keys that can be used to subset the data rows. The keys values are the rsid's.

`select(x, keys = NULL, cols = NULL, ...)`: Returns a subset of data defined by the character vectors `keys` and `cols`. If no keys are supplied, all rows are returned. If no cols are supplied, all columns are returned. For column descriptions see `?SIFTDbColumns`.

## Author(s)

Valerie Obenchain <[vobencha@fhcrc.org](mailto:vobencha@fhcrc.org)>

## References

SIFT Home: <http://sift.jcvi.org/>

Kumar P, Henikoff S, Ng PC. Predicting the effects of coding non-synonymous variants on protein function using the SIFT algorithm. *Nat Protoc.* 2009;4(7):1073-81

Ng PC, Henikoff S. Predicting the Effects of Amino Acid Substitutions on Protein Function *Annu Rev Genomics Hum Genet.* 2006;7:61-80.

Ng PC, Henikoff S. SIFT: predicting amino acid changes that affect protein function. *Nucleic Acids Res.* 2003 Jul 1;31(13):3812-4.

## Examples

```
library(SIFT.Hsapiens.dbSNP132)

## metadata
metadata(SIFT.Hsapiens.dbSNP132)

## available rsid's
head(keys(SIFT.Hsapiens.dbSNP132))

## for column descriptions see ?SIFTDbColumns
cols(SIFT.Hsapiens.dbSNP132)

## subset on keys and cols
rsids <- c("rs2142947", "rs17970171", "rs8692231", "rs3026284")
subst <- c("RSID", "PREDICTION", "SCORE")
select(SIFT.Hsapiens.dbSNP132, keys=rsids, cols=subst)
select(SIFT.Hsapiens.dbSNP132, keys=rsids[1:2])
```

---

SIFTDbColumns

*SIFTDb Columns*

---

## Description

Description of the SIFT Sqlite Database Columns

## Column descriptions

These column names are displayed when `cols` is called on a SIFTDb object.

- **RSID** : rsid
- **PROTEINID** : NCBI RefSeq protein ID
- **AACHANGE** : amino acid substitution; reference aa is preceding, followed by the position and finally the snp aa
- **METHOD** : method of obtaining related sequences using PSI-BLAST
- **AA** : either the reference or snp residue amino acid
- **PREDICTION** : SIFT prediction
- **SCORE** : SIFT score (range 0 to 1)
  - **TOLERATED** : score is greater than 0.05
  - **DAMAGING** : score is less than or equal to 0.05
  - **NOT SCORED** : no prediction is made if there are less than 2 homologous sequences that have an amino acid at the position of the given SNP or if the SIFT prediction is not available
- **MEDIAN** : diversity measurement of the sequences used for prediction (range 0 to 4.32)
- **POSITIONSEQS** : number of sequences with an amino acide at the position of prediction
- **TOTALSEQS** : total number of sequences in alignment

## Author(s)

Valerie Obenchain <[vobencha@fhcrc.org](mailto:vobencha@fhcrc.org)>

## See Also

?SIFTDb

---

summarizeVariants	<i>Summarize variants by sample</i>
-------------------	-------------------------------------

---

## Description

Variants in a VCF file are overlapped with an annotation region and summarized by sample. Genotype information in the VCF is used to determine which samples express each variant.

## Usage

```
## S4 method for signature 'TranscriptDb,VCF,CodingVariants'
summarizeVariants(query, subject, mode, ...)
## S4 method for signature 'TranscriptDb,VCF,FiveUTRVariants'
summarizeVariants(query, subject, mode, ...)
## S4 method for signature 'TranscriptDb,VCF,ThreeUTRVariants'
summarizeVariants(query, subject, mode, ...)
## S4 method for signature 'TranscriptDb,VCF,SpliceSiteVariants'
summarizeVariants(query, subject, mode, ...)
## S4 method for signature 'TranscriptDb,VCF,IntronVariants'
summarizeVariants(query, subject, mode, ...)
## S4 method for signature 'TranscriptDb,VCF,PromoterVariants'
summarizeVariants(query, subject, mode, ...)
## S4 method for signature 'GRangesList,VCF,VariantType'
summarizeVariants(query, subject, mode, ...)
## S4 method for signature 'GRangesList,VCF,function'
summarizeVariants(query, subject, mode, ...)
```

## Arguments

query	A <a href="#">TranscriptDb</a> or <a href="#">GRangesList</a> object that serves as the annotation. GFF files can be converted to <a href="#">TranscriptDb</a> objects with <code>makeTranscriptDbFromGFF()</code> in the <code>GenomicFeatures</code> package.
subject	A <a href="#">VCF</a> object containing the variants.
mode	mode can be a <code>VariantType</code> class or the name of a function. When mode is a <code>VariantType</code> class, counting is done with <code>locateVariants</code> and counts are summarized transcript-by-sample. Supported <code>VariantType</code> classes include <code>CodingVariants</code> , <code>IntronVariants</code> , <code>FiveUTRVariants</code> , <code>ThreeUTRVariants</code> , <code>SpliceSiteVariants</code> or <code>PromoterVariants</code> . <code>AllVariants()</code> and <code>IntergenicVariants</code> are not supported. See <code>?locateVariants</code> for more detail on the variant classes. mode can also be the name of any counting function that outputs a <code>Hits</code> object. Variants will be summarized by the length of the <code>GRangesList</code> annotation (i.e., 'length-of-GRangesList'-by-sample).
...	Additional arguments passed to methods such as <b>ignore.strand</b> A logical indicating if strand should be ignored when performing overlaps.

## Details

summarizeVariants uses the genotype information in a VCF file to determine which samples are positive for each variant. Variants are overlapped with the annotation and the counts are summarized annotation-by-sample. If the annotation is a GRangesList of transcripts, the count matrix will be transcripts-by-sample. If the GRangesList is genes, the count matrix will be gene-by-sample.

- Counting with locateVariants() :  
Variant counts are always summarized transcript-by-sample. When query is a GRangesList, it must be compatible with the VariantType-class given as the mode argument. The list below specifies the appropriate GRangesList for each mode.  
**CodingVariants** : coding (CDS) by transcript  
**IntronVariants** : introns by transcript  
**FiveUTRVariants** : five prime UTR by transcript  
**ThreeUTRVariants** : three prime UTR by transcript  
**SpliceSiteVariants** : introns by transcript  
**PromoterVariants** : list of transcripts  
When query is a TranscriptDb, the appropriate region-by-transcript GRangesList listed above is extracted internally and used as the annotation.
- Counting with a user-supplied function :  
subject must be a GRangesList and mode must be the name of a function. The count function must take 'query' and 'subject' arguments and return a Hits object. Counts are summarized by the outer list elements of the GRangesList.

## Value

A SummarizedExperiment object with count summaries in the assays slot. The rowData contains the annotation used for counting. Information in colData and exptData are taken from the VCF file.

## Author(s)

Valerie Obenchain <vobencha@fhcrc.org>

## See Also

[readVcf](#), [predictCoding](#) [locateVariants](#) [summarizeVariants](#),

## Examples

```
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene

## Read variants from VCF and adjust seqlevels
fl <- system.file("extdata", "chr22.vcf.gz", package="VariantAnnotation")
vcf <- readVcf(fl, "hg19")
newnames <- paste("chr", seqlevels(vcf), sep="")
names(newnames) <- seqlevels(vcf)
vcf <- renameSeqlevels(vcf, newnames)
## Confirm seqlevels
intersect(seqlevels(vcf), seqlevels(txdb))

## -----
```

```

## Counting with locateVariants()
## -----
## TranscriptDb as the 'query'
coding1 <- summarizeVariants(txdb, vcf, CodingVariants())
colSums(assays(coding1)$counts)

## GRangesList as the 'query'
cdsbytx <- cdsBy(txdb, "tx")
coding2 <- summarizeVariants(cdsbytx, vcf, CodingVariants())

stopifnot(identical(assays(coding1)$counts, assays(coding2)$counts))

## Promoter region variants summarized by transcript
tx <- transcripts(txdb)
txlst <- splitAsList(tx, seq_len(length(tx)))
promoter <- summarizeVariants(txlst, vcf,
                             PromoterVariants(upstream=100, downstream=10))
colSums(assays(promoter)$counts)

## -----
## Counting with findOverlaps()
## -----

## Summarize all variants by transcript
allvariants <- summarizeVariants(txlst, vcf, findOverlaps)
colSums(assays(allvariants)$counts)

```

VAFilter-class

*VAFilter objects - under construction***Description**

Objects of the VAFilter class are functions for filtering variants. A VAFilter instance applied to a GRanges returns either a subset of the records that passed the filter or a VAFilterResult object. The VAFilterResult is a logical vector indicating which records passed the filter and contains summary statistics on the records that passed.

**Arguments**

fun	An object of class function to be used as a filter. fun must accept a single named argument x, and is expected to return a logical vector such that x[fun(x)] selects only those elements of x satisfying the conditions of fun
name	A character(1) object to be used as the name of the filter. The name is useful for debugging and reference.
filt	A VAFilter object, to be used with additional arguments to create a composite filter.
.name	An optional character(1) object used to over-ride the name applied to default filters.
dbSNP	The character(0) name of the dbSNP package to be used. The default (character(0)) performs no filtering
txdb	The TranscriptDb object used to identifying gene regions.

region	A character(1) object specifying the region on which to filter the results. Possible values are 'coding', 'intron', '3'UTR', '5'UTR', 'intergenic', 'noGeneMatch' and 'unknown'. See ?locateVariants for regions descriptions.
...	Additional arguments to methods.

### Details

A VAFilter is constructed by using one of the built-in filters, dbSNPFilter or regionFilter, or a user can create their own with vaFilter. Once the filter is created it can be applied to a GRanges object. The format of the output is dictated by the optionalsubset argument. If subset=TRUE (the default) a GRanges is returned containing only the records that passed the filter. If subset=FALSE a VAFilterResult object is returned. This object contains a logical vector indicating which records passed the filter and summary statistics on the records that passed.

### Slots

**.Data:** Object of class "function" taking a single named argument x corresponding to the GRanges object that the filter will be applied to. The return value of the filter function is expected to be a logical vector that can be used to subset x to include those elements of x satisfying the filter.

**name:** Object of class "ScalarCharacter" representing the name of the filter. The name is useful for suggesting the purpose of the filter, and for debugging failed filters.

### Constructors

Built-in filters :

dbSNPFilter(dbSNP=character(0), name="dbSNPFilter"): Overlaps the records with a range width of 1 with snp locations in the specified SNPllocs package.

regionFilter(txdb, region="coding", name="regionFilter"): Records will be filtered by the name of the region supplied. Possible regions are the same as those in the Location output from locateVariants: 'coding', 'intronic', '3UTR', '5UTR', 'intergenic', 'noGeneMatch' and 'unknown'. This regions is specified when calling the filter on a GRanges object.

Combining filters :

compose(filt, ..., .name): The compose function constructs a new filter from one or more existing filters. The result is a filter that returns records that pass the criteria for all filters. If a name is not provided for the filter, a name will be constructed by joining the names of all component filters separated by " o ".

User defined filters : A user can construct their own filter using the vaFilter function. The fun argument must be a function accepting a single argument x and returning a logical vector indicating which records passed the filter.

vaFilter(fun="VAFilter", name=NA\_character\_, ...): Returns the function representing the underlying filter. This is primarily for viewing the filter function.

vaFilter(fun="missing", name=NA\_character\_, ...): Creates a default filter that returns a vector of TRUE values with length equal to length(x).

### Other Methods

**name** signature(x = "VAFilter"): Return, as a ScalarCharacter, the name of the function.

**show** signature(object = "VAFilter"): display a brief summary of the filter



**Author(s)**

Valerie Obenchain <vobencha@fhcrc.org>

**See Also**

[VAFilterResult](#)

**Examples**

```
# data(variants)
#
# ## Create a filter to select only records in introns :
# library(TxDb.Hsapiens.UCSC.hg19.knownGene)
# txdb19 <- TxDb.Hsapiens.UCSC.hg19.knownGene
# regionFilt <- regionFilter(txdb19, region="intron")
# ## View the filter with vaFilter()
# vaFilter(regionFilt)
# regionFilt(variants, subset=FALSE)
#
# ## Create a filter to identify variants present in dbSNP :
# library("SNPlocs.Hsapiens.dbSNP.20110815")
# snpFilt <- dbSNPFilter("SNPlocs.Hsapiens.dbSNP.20110815")
# ## Modify the seqlevels to match the SNPlocs package
# variants <- renameSeqlevels(variants, c("chr16"="ch16"))
# ## Apply the filter to chromosome 16 snps :
# ## Optional subset argument is TRUE (default)
# snpFilt(variants[seqnames(variants) == "ch16"], subset=TRUE)
```

---

VAFilterResult-class      *"VAFilterResult" for VAFilter output and statistics*

---

**Description**

Objects of this class are logical vectors indicating records passing the applied filter, with an associated data frame summarizing the name, input number of records, records passing filter, and logical operation used for all filters in which the result participated.

**Usage**

```
VAFilterResult(data = GRanges(), x = logical(), name = NA_character_,
  subset = TRUE, input = length(x), passing = sum(x), op = NA_character_)
## S4 method for signature 'VAFilterResult,VAFilterResult'
Logic(e1, e2)
## S4 method for signature 'VAFilterResult'
name(x, ...)
stats(x, ...)
## S4 method for signature 'VAFilterResult'
show(object)
```

**Arguments**

data	A linkS4class{GRanges} of the data to be filtered.
x, object, e1, e2	For VAFilterResult, logical() indicating records that passed filter or, for others, an instance of VAFilterResult class.
name	character() indicating the name by which the filter is to be referred. Internally, name, input, passing, and op may all be vectors representing columns of a data.frame summarizing the application of successive filters.
subset	logical() when TRUE, the function returns a subset of the data that passed the filter as a linkS4class{GRanges} object. The name and stats information are in the metadata slot. When FALSE, an object of VAFilterResult is returned.
input	integer() indicating the length of the original input.
passing	integer() indicating the number of records passing the filter.
op	character() indicating the logical operation, if any, associated with this filter.
...	Additional arguments, unused in methods documented on this page.

**Objects from the Class**

Objects can be created through [VAFilterResult](#), but these are automatically created by the application of [vaFilter](#) instances.

**Slots**

**.Data:** Object of class "logical" indicating records that passed the filter.

**name:** Object of class "ScalarCharacter" representing the name of the filter whose results are summarized. The name is either the actual name of the filter, or a combination of filter names and logical operations when the outcome results from application of several filters in a single logical expression.

**stats:** Object of class "data.frame" summarizing the name, input number of records, records passing filter, and logical operation used for all filters in which the result participated. The data.frame rows correspond either to single filters, or to logical combinations of filters.

**Methods**

**Logic** signature(e1 = "VAFilterResult", e2 = "VAFilterResult"): logic operations on filters.

**!** signature(x = "VAFilterResult"): Negate the outcome of the current filter results

**name** signature(x = "VAFilterResult"): The name of the filter that the results are based on.

**stats** signature(x = "VAFilterResult"): a data.frame as described in the 'Slots' section of this page.

**show** signature(object = "VAFilterResult"): summary of filter results.

**Author(s)**

Valerie Obenchain <[vobencha@fhcrc.org](mailto:vobencha@fhcrc.org)>

**See Also**

[VAFilter](#)

**Examples**

```
## see ?VAFilter
```

---

VariantType-class	<i>VariantType subclasses</i>
-------------------	-------------------------------

---

**Description**

VariantType subclasses specify the type of variant to be located with locateVariants.

**Usage**

```
CodingVariants()
IntronVariants()
FiveUTRVariants()
ThreeUTRVariants()
SpliceSiteVariants()
IntergenicVariants()
PromoterVariants(upstream = 2000, downstream = 200)
AllVariants(upstream = 2000, downstream = 200)
```

**Arguments**

upstream	A numeric indicating the number of base pairs upstream of the 5'-end.
downstream	A numeric indicating the number of base pairs downstream of the 3'-end.

**Details**

VariantType is a virtual class inherited by the CodingVariants, IntronVariants, FiveUTRVariants, ThreeUTRVariants, SpliceSiteVariants, IntergenicVariants and AllVariants subclasses.

The subclasses are used as the region argument to locateVariants. They designate the type of variant (i.e., region of the annotation to match) when calling locateVariants.

The majority of subclasses have no slots and require no arguments for an instance to be created. The two exceptions are PromoterVariants and AllVariants. These class have slots for the number of base pairs upstream of the 5'-end and downstream of the 3'-end of the transcript region.

**Accessors**

In the following code, x is a PromoterVariants or a AllVariants object.

```
upstream(x), upstream(x) <- value: Gets or sets the number of base pairs defining a range
upstream of the 5' end (excludes 5' start value).

downstream(x), downstream(x) <- value: Gets or sets the number of base pairs defining a range
downstream of the 3' end (excludes 3' end value).
```

**Author(s)**

Valerie Obenchain <vobencha@fhcrc.org>

**Examples**

```
CodingVariants()
SpliceSiteVariants()
PromoterVariants(upstream=1000, downstream=10000)
```

---

VAUtil-class

*".VAUtil" and potentially other related classes - in progress*


---

**Description**

These classes provide important utility functions in the **VariantAnnotation** package, but may occasionally be seen by the user and are documented here for that reason.

**Objects from the Class**

Utility classes include:

- .VAUtil-class a virtual base class from which all utility classes are derived.

**Author(s)**

Valerie Obenchain <vobencha@fhcrc.org>

**Examples**

```
getClass(".VAUtil", where=getNamespace("VariantAnnotation"))
```

---

VCF-class

*VCF instances*


---

**Description**

The VCF class is an extension of the [SummarizedExperiment](#)-class with two additional slots, fixed and info.

**Details**

The VCF class is designed to hold data from a Variant Call Format (VCF) file. The class extends [SummarizedExperiment](#) with the addition of two slots, fixed and info. See `?SummarizedExperiment` for a detailed description of the inherited slots.

Slots unique to the VCF class,

fixed A [DataFrame](#) containing information from the REF, ALT, QUAL and FILTER fields from a VCF file.

info A [DataFrame](#) containing information from the INFO fields from a VCF file.

Slots inherited from the [SummarizedExperiment](#) class,

exptData A [SimpleList](#)-class instance containing the file header or other information about the overall experiment.

`rowData` A [GRanges](#)-class instance defining the variant ranges and associated metadata columns of REF, ALT, QUAL and FILTER.

`colData` A [DataFrame](#)-class instance describing the samples and associated metadata.

`geno` The assays slot from [SummarizedExperiment](#) has been renamed as `geno` for the VCF class. This slot contains the genotype information immediately following the FORMAT field in a VCF file. Each element of the list or [SimpleList](#) is a matrix or array.

## Extends

Directly extends class [SummarizedExperiment](#).

## Constructor

```
VCF(rowData = GRanges(), colData = DataFrame(),          exptData = SimpleList(), fixed = DataFrame(),
```

## Accessors

In the following code snippets `x` is a VCF object. The `info` and `fixed` accessors returns the data as `elementMetadata` column(s) of the `rowData` [GRanges](#) object. The `geno` accessor returns a [SimpleList](#).

```
ref(x), ref(x) <- value: Returns or sets the reference allele data from the REF column of the VCF file. value must be a DNASTringSet.
```

```
alt(x), alt(x) <- value: Returns or sets the alternate allele data from the ALT column of the VCF file. value can be a DNASTringSetList or a CharacterList.
```

```
qual(x), qual(x) <- value: Returns or sets the quality scores from the QUAL column of the VCF file. value must be an integer(1L).
```

```
filt(x), filt(x) <- value: Returns or sets the filter data from the FILTER column of the VCF file. value must be a character(1L).
```

```
fixed(x), fixed(x) <- value: The 'getter' returns the GRanges in the rowData slot with the REF, ALT, QUAL, and FILTER fields as metadata columns. If the data were read in with readVcf() the GRanges will have an additional metadata column called paramRangeID. This column distinguishes ranges when a subset of data have been read in from a VCF file.
```

The 'setter' sets the values in the `info` slot of the VCF. value must be a [DataFrame](#).

```
info(x), info(x) <- value: The 'getter' returns the GRanges in the rowData slot with the INFO variables from the VCF file as metadata columns. If the data were read in with readVcf() the GRanges will have an additional metadata column called paramRangeID. This column distinguishes ranges when a subset of data have been read in from a VCF file.
```

The 'setter' sets the values in the `fixed` slot of the VCF. value must be a [DataFrame](#). If no `info` fields are present, the `rowData` [GRanges](#) will be returned with no metadata columns.

```
geno(x), geno(x) <- value: Returns or sets geno data. Contains the genotype information from the samples in a VCF file. value must be a SimpleList of matrices or arrays. An optional withDimnames argument controls the return of dimnames (default = TRUE).
```

```
geno(x)[[i]], geno(x)[[i]] <- value: Returns or sets elements of geno. value can be a matrix, or array.
```

```
exptData(x), exptData(x) <- value: Returns or sets exptData data. Contains the header information from a VCF file as well as any other experiment-specific information. value must be a SimpleList.
```

`rowData(x)`, `rowData(x) <- value`: Returns or sets `rowData` data. Contains a `GRanges` constructed from the `CHROM`, `POS` and `ID` fields of the VCF file. The `ID`'s serve as the rownames; if they are `NULL`, rownames are constructed from `CHROM:POS`. `value` must be a `GRanges` with names representing the `ID`'s in the VCF file.

`colData(x)`, `colData(x) <- value`: Returns or sets `colData` data. Contains a `DataFrame` of sample-specific information. Each row represents a sample in the VCF file. `value` must be a `DataFrame` with rownames representing the samples in the VCF file.

### Subsetting

In the following code snippets `x` is a VCF object.

`x[i, j]`, `x[i, j] <- value`: Gets or sets rows `i` and columns `j`. `i` and `j` can be integer or logical vectors. `value` is a replacement VCF object.

### Other methods

`genome(x)`: Extract the genome information from the `GRanges` in the `rowData` slot.

`seqlevels(x)`: Extract the `seqlevels` from the `GRanges` in the `rowData` slot.

`strand(x)`: Extract the strand from the `GRanges` in the `rowData` slot.

`renameSeqlevels(x, value)`: Rename the `seqlevels` in the `GRanges` in the `rowData` slot of the VCF object. `value` is a named character vector where the names are the old `seqlevels` and the values are the new.

`keepSeqlevels(x, value)`: Subset the `GRanges` in the `rowData` slot of the VCF object. `value` is a character vector of `seqlevels` to keep.

### Arguments

**geno** A list or `SimpleList` of matrix elements, or a matrix containing the genotype information from a VCF file. If present, these data immediately follow the `FORMAT` field in the VCF.

Each element of the list must have the same dimensions, and dimension names (if present) must be consistent across elements and with the row names of `rowData`, `colData`.

**info** A `DataFrame` of data from the `INFO` field of a VCF file. The number of rows must match that in the `rowData` object.

**fixed** A `DataFrame` of `REF`, `ALT`, `QUAL` and `FILTER` fields from a VCF file. The number of rows must match that of the `rowData` object.

**rowData** A `GRanges` instance describing the ranges of interest. Row names, if present, become the row names of the VCF. The length of the `GRanges` must equal the number of rows of the matrices in `geno`.

**colData** A `DataFrame` describing the samples. Row names, if present, become the column names of the VCF.

**exptData** A `SimpleList` describing the header of the VCF file or additional information for the overall experiment.

**...** Additional arguments passed to methods.

**withDimnames** A `logical(1)`, indicating whether `dimnames` should be applied to extracted assay elements. Applicable to the `geno` accessor only.

**verbose** A `logical(1)` indicating whether messages about data coercion during construction should be printed.

**Author(s)**

Valerie Obenchain <vobencha@fhcrc.org>

**See Also**

[GRanges](#), [DataFrame](#), [SimpleList](#), [SummarizedExperiment](#), [readVcf](#), [writeVcf](#)

**Examples**

```
fl <- system.file("extdata", "structural.vcf", package="VariantAnnotation")
vcf <- readVcf(fl, genome="hg19")

## Extract individual 'fixed' fields with ref, alt, qual and filt accessors
qual(vcf)
ref(vcf)

## Extract 'fixed', 'info' and 'geno' fields
head(fixed(vcf))
head(info(vcf))
geno(vcf)
identical(geno(vcf)$DP, geno(vcf)[[3]])

## Rename and subset the seqlevels in the rowData GRanges object :
## The seqlevels (chromosome names) of the vcf are numbers instead of
## being preceded by "chr*" or "ch*"
seqlevels(vcf)

## Add "chr" in front of the seqlevels using renameSeqlevels()
newnames <- paste("chr", seqlevels(vcf), sep="")
names(newnames) <- seqlevels(vcf)
vcf <- renameSeqlevels(vcf, newnames)
seqlevels(vcf)

## To subset on seqlevels use the keepSeqlevels() helper function
vcf_subset <- keepSeqlevels(vcf, "chr2")
seqlevels(vcf_subset)
```

---

VCFHeader-class

*VCFHeader instances*

---

**Description**

The VCFHeader class holds Variant Call Format (VCF) file header information and is produced from a call to `scanVcfHeader`.

**Details**

The VCFHeader class is holds header information from a from VCF file.

Slots :

reference `character()` vector

sample `character()` vector

header [DataFrameList-class](#)

**Constructor**

VCFHeader(reference = character(), samples = character(), header = DataFrameList(), ...)

**Accessors**

In the following code snippets x is a VCFHeader object.

samples(x): Returns a character() vector of names of samples.

header(x): Returns all information in the header slot which includes meta, info and geno if present.

meta(x): Returns a DataFrameList of meta information. This includes any information represented as simple key=value pairs in the VCF file header.

fixed(x): Returns a DataFrameList of information pertaining to any of 'REF', 'ALT', 'FILTER' and 'QUAL'.

info(x): Returns a DataFrame of 'INFO' information.

geno(x): Returns a DataFrame of 'FORMAT' information.

reference(x): Returns a character() vector with names of reference sequences. Not relevant for scanVcfHeader.

**Arguments**

**reference** A character() vector of sequences.

**sample** A character() vector of sample names.

**header** A DataFrameList of parsed header lines (preceded by "##") present in the VCF file.

... Additional arguments passed to methods.

**Author(s)**

Valerie Obenchain <vobencha@fhcrc.org>

**See Also**

[scanVcfHeader](#), [DataFrameList](#)

**Examples**

```
fl <- system.file("extdata", "structural.vcf", package="VariantAnnotation")
hdr <- scanVcfHeader(fl)

fixed(hdr)
info(hdr)
geno(hdr)
```



---

writeVcf	<i>Write VCF files</i>
----------	------------------------

---

**Description**

Write Variant Call Format (VCF) files to disk

**Usage**

```
## S4 method for signature 'VCF,character'  
writeVcf(obj, filename, index = FALSE, ...)
```

**Arguments**

obj	Object containing data to be written out. At present only accepts <a href="#">VCF</a> .
filename	The character() name of the VCF file to be written out.
index	Whether to bgzip the output file and generate a tabix index.
...	Additional arguments, passed to methods.

**Details**

A VCF file can be written out from data in a VCF file. More general methods to write out from other objects may be added in the future.

**Value**

VCF file

**Author(s)**

Valerie Obenchain <[vobencha@fhcrc.org](mailto:vobencha@fhcrc.org)> and Michael Lawrence

**References**

<http://vcftools.sourceforge.net/specs.html> outlines the VCF specification.

<http://samtools.sourceforge.net/mpileup.shtml> contains information on the portion of the specification implemented by bcftools.

<http://samtools.sourceforge.net/> provides information on samtools.

**See Also**

[readVcf](#)

**Examples**

```
fl <- system.file("extdata", "ex2.vcf", package="VariantAnnotation")

out1.vcf <- tempfile()
out2.vcf <- tempfile()
in1 <- readVcf(fl, "hg19")
writeVcf(in1, out1.vcf)
in2 <- readVcf(out1.vcf, "hg19")
writeVcf(in2, out2.vcf)
in3 <- readVcf(out2.vcf, "hg19")

identical(in2, in3)
```

# Index

- !,VAFilterResult-method  
(VAFilterResult-class), 33
- \*Topic **classes**
  - PolyPhenDb-class, 8
  - PolyPhenDbColumns, 10
  - ScanVcfParam-class, 24
  - SIFTDb-class, 26
  - SIFTDbColumns, 28
  - VAFilter-class, 31
  - VAFilterResult-class, 33
  - VAUtil-class, 36
- \*Topic **manip**
  - getTranscriptSeqs, 2
  - MatrixToSnpMatrix, 6
  - readVcf, 15
  - readVcfLongForm, 18
  - refLocsToLocalLocs, 20
  - scanVcf, 22
  - writeVcf, 41
- \*Topic **methods**
  - getTranscriptSeqs, 2
  - locateVariants, 3
  - PolyPhenDb-class, 8
  - PolyPhenDbColumns, 10
  - predictCoding, 12
  - SIFTDb-class, 26
  - SIFTDbColumns, 28
  - summarizeVariants, 29
- .VAUtil-class (VAUtil-class), 36
- [,VCF,ANY,ANY-method (VCF-class), 36
- [,VCF-method (VCF-class), 36
- [<-,VCF,ANY,ANY,VCF-method  
(VCF-class), 36
  
- AllVariants (VariantType-class), 35
- AllVariants-class (VariantType-class), 35
- alt (VCF-class), 36
- alt,VCF-method (VCF-class), 36
- alt<- (VCF-class), 36
- alt<-,VCF,CharacterList-method  
(VCF-class), 36
- alt<-,VCF,DNAStringSetList-method  
(VCF-class), 36
  
- BcfFile, 24
- BSgenome, 2, 13
  
- class:PolyPhenDb (PolyPhenDb-class), 8
- class:SIFTDb (SIFTDb-class), 26
- class:VAFilter (VAFilter-class), 31
- CodingVariants (VariantType-class), 35
- CodingVariants-class (VariantType-class),  
35
- cols,PolyPhenDb-method  
(PolyPhenDb-class), 8
- cols,SIFTDb-method (SIFTDb-class), 26
- compose (VAFilter-class), 31
- contig (VCFHeader-class), 39
- contig,VCFHeader-method  
(VCFHeader-class), 39
  
- DataFrame, 36, 37, 39
- DataFrameList, 39, 40
- dbSNPFilter (VAFilter-class), 31
- DNAStringSet, 2, 13
- downstream (VariantType-class), 35
- downstream,AllVariants-method  
(VariantType-class), 35
- downstream,PromoterVariants-method  
(VariantType-class), 35
- downstream<- (VariantType-class), 35
- downstream<- ,AllVariants-method  
(VariantType-class), 35
- downstream<- ,PromoterVariants-method  
(VariantType-class), 35
- duplicateRSID (PolyPhenDb-class), 8
  
- extractTranscriptsFromGenome, 3, 21
  
- FaFile, 2, 13
- filt (VCF-class), 36
- filt,VCF-method (VCF-class), 36
- filt<- (VCF-class), 36
- filt<- ,VCF,character-method (VCF-class),  
36
- FiveUTRVariants (VariantType-class), 35
- FiveUTRVariants-class  
(VariantType-class), 35

- fixed (VCF-class), 36
- fixed,VCF-method (VCF-class), 36
- fixed,VCFHeader-method (VCFHeader-class), 39
- fixed<- (VCF-class), 36
- fixed<-,VCF,DataFrame-method (VCF-class), 36
  
- geno (VCF-class), 36
- geno,VCF-method (VCF-class), 36
- geno,VCFHeader-method (VCFHeader-class), 39
- geno<- (VCF-class), 36
- geno<-,VCF,character,matrix-method (VCF-class), 36
- geno<-,VCF,missing,SimpleList-method (VCF-class), 36
- geno<-,VCF,numeric,matrix-method (VCF-class), 36
- genome,VCF-method (VCF-class), 36
- getSeq, 3
- getTranscriptSeqs, 2, 14
- getTranscriptSeqs,GRanges,FaFile-method (getTranscriptSeqs), 2
- getTranscriptSeqs,GRangesList,BSgenome-method (getTranscriptSeqs), 2
- getTranscriptSeqs,GRangesList,FaFile-method (getTranscriptSeqs), 2
- GRanges, 3, 12, 13, 15, 19, 21, 25, 31, 32, 37, 39
- GRangesList, 2, 21
  
- header (VCFHeader-class), 39
- header,VCFHeader-method (VCFHeader-class), 39
- Hits, 3
  
- indexTabix, 17
- info (VCF-class), 36
- info,VCF-method (VCF-class), 36
- info,VCFHeader-method (VCFHeader-class), 39
- info<- (VCF-class), 36
- info<-,VCF,DataFrame-method (VCF-class), 36
- IntergenicVariants (VariantType-class), 35
- IntergenicVariants-class (VariantType-class), 35
- IntronVariants (VariantType-class), 35
- IntronVariants-class (VariantType-class), 35
  
- keepSeqlevels,VCF,character-method (VCF-class), 36
  
- keys,PolyPhenDb-method (PolyPhenDb-class), 8
- keys,SIFTDb-method (SIFTDb-class), 26
  
- locateVariants, 3, 14, 30
- locateVariants,GRanges,GRangesList,AllVariants-method (locateVariants), 3
- locateVariants,GRanges,GRangesList,CodingVariants-method (locateVariants), 3
- locateVariants,GRanges,GRangesList,FiveUTRVariants-method (locateVariants), 3
- locateVariants,GRanges,GRangesList,IntergenicVariants-method (locateVariants), 3
- locateVariants,GRanges,GRangesList,IntronVariants-method (locateVariants), 3
- locateVariants,GRanges,GRangesList,PromoterVariants-method (locateVariants), 3
- locateVariants,GRanges,GRangesList,SpliceSiteVariants-method (locateVariants), 3
- locateVariants,GRanges,GRangesList,ThreeUTRVariants-method (locateVariants), 3
- locateVariants,GRanges,GRangesList,VariantType-method (locateVariants), 3
- locateVariants,GRanges,TranscriptDb,AllVariants-method (locateVariants), 3
- locateVariants,GRanges,TranscriptDb,CodingVariants-method (locateVariants), 3
- locateVariants,GRanges,TranscriptDb,FiveUTRVariants-method (locateVariants), 3
- locateVariants,GRanges,TranscriptDb,IntergenicVariants-method (locateVariants), 3
- locateVariants,GRanges,TranscriptDb,IntronVariants-method (locateVariants), 3
- locateVariants,GRanges,TranscriptDb,PromoterVariants-method (locateVariants), 3
- locateVariants,GRanges,TranscriptDb,SpliceSiteVariants-method (locateVariants), 3
- locateVariants,GRanges,TranscriptDb,ThreeUTRVariants-method (locateVariants), 3
- locateVariants,GRanges,TranscriptDb,VariantType-method (locateVariants), 3
- locateVariants,Ranges,GRangesList,VariantType-method (locateVariants), 3
- locateVariants,Ranges,TranscriptDb,VariantType-method (locateVariants), 3
- locateVariants,VCF,GRangesList,VariantType-method (locateVariants), 3
- locateVariants,VCF,TranscriptDb,VariantType-method (locateVariants), 3
- Logic,VAFilterResult,VAFilterResult-method (VAFilterResult-class), 33
  
- map, 21

- MatrixToSnpMatrix, 6
- MatrixToSnpMatrix,matrix,DNAStringSet,DNAStringSetList-method (MatrixToSnpMatrix), 6
- meta (VCFHeader-class), 39
- meta,VCFHeader-method (VCFHeader-class), 39
- metadata,PolyPhenDb-method (PolyPhenDb-class), 8
- metadata,SIFTDb-method (SIFTDb-class), 26
- name (VAFilter-class), 31
- name,VAFilter-method (VAFilter-class), 31
- name,VAFilterResult-method (VAFilterResult-class), 33
- PolyPhen (PolyPhenDb-class), 8
- PolyPhenDb (PolyPhenDb-class), 8
- PolyPhenDb-class, 8
- PolyPhenDbColumns, 10
- predictCoding, 3, 5, 12, 30
- predictCoding,GRanges,TranscriptDb,ANY,DNAStringSet-method (predictCoding), 12
- predictCoding,Ranges,TranscriptDb,ANY,DNAStringSet-method (predictCoding), 12
- predictCoding,VCF,TranscriptDb,ANY,missing-method (predictCoding), 12
- PromoterVariants (VariantType-class), 35
- PromoterVariants-class (VariantType-class), 35
- qual (VCF-class), 36
- qual,VCF-method (VCF-class), 36
- qual<- (VCF-class), 36
- qual<-,VCF,integer-method (VCF-class), 36
- RangedData, 25
- Ranges, 3, 12
- RangesList, 25
- readVcf, 5, 7, 14, 15, 20, 24, 25, 30, 39, 41
- readVcf,character,character,missing-method (readVcf), 15
- readVcf,character,character,ScanVcfParam-method (readVcf), 15
- readVcf,character,missing,missing-method (readVcf), 15
- readVcf,TabixFile,character,GRanges-method (readVcf), 15
- readVcf,TabixFile,character,missing-method (readVcf), 15
- readVcf,TabixFile,character,RangedData-method (readVcf), 15
- readVcf,TabixFile,character,RangesList-method (readVcf), 15
- readVcf,TabixFile,character,ScanVcfParam-method (readVcf), 15
- readVcfLongForm, 18
- readVcfLongForm,character,character,missing-method (readVcfLongForm), 18
- readVcfLongForm,character,character,ScanVcfParam-method (readVcfLongForm), 18
- readVcfLongForm,character,missing,missing-method (readVcfLongForm), 18
- readVcfLongForm,TabixFile,character,GRanges-method (readVcfLongForm), 18
- readVcfLongForm,TabixFile,character,missing-method (readVcfLongForm), 18
- readVcfLongForm,TabixFile,character,RangedData-method (readVcfLongForm), 18
- readVcfLongForm,TabixFile,character,RangesList-method (readVcfLongForm), 18
- readVcfLongForm,TabixFile,character,ScanVcfParam-method (readVcfLongForm), 18
- ref (VCF-class), 36
- ref,VCF-method (VCF-class), 36
- ref<- (VCF-class), 36
- ref<-,VCF,DNAStringSet-method (VCF-class), 36
- reference (VCFHeader-class), 39
- reference,VCFHeader-method (VCFHeader-class), 39
- refLocsToLocalLocs, 14, 20
- refLocsToLocalLocs,GRanges,missing,GRangesList-method (refLocsToLocalLocs), 20
- refLocsToLocalLocs,GRanges,TranscriptDb,missing-method (refLocsToLocalLocs), 20
- regionFilter (VAFilter-class), 31
- renameSeqlevels,VCF,character-method (VCF-class), 36
- samples (VCFHeader-class), 39
- samples,VCFHeader-method (VCFHeader-class), 39
- scanBcf, 17
- scanTabix, 17
- scanVcf, 16, 22
- scanVcf,character,missing-method (scanVcf), 22
- scanVcf,character,ScanVcfParam-method (scanVcf), 22
- scanVcf,connection,missing-method (scanVcf), 22
- scanVcf,TabixFile,GRanges-method (scanVcf), 22

scanVcf,TabixFile,missing-method  
     (scanVcf), 22  
 scanVcf,TabixFile,RangedData-method  
     (scanVcf), 22  
 scanVcf,TabixFile,RangesList-method  
     (scanVcf), 22  
 scanVcf,TabixFile,ScanVcfParam-method  
     (scanVcf), 22  
 scanVcfHeader, 40  
 scanVcfHeader (scanVcf), 22  
 scanVcfHeader,character-method (scanVcf),  
     22  
 scanVcfHeader,TabixFile-method (scanVcf),  
     22  
 ScanVcfParam, 15, 19, 23  
 ScanVcfParam (ScanVcfParam-class), 24  
 ScanVcfParam,ANY-method  
     (ScanVcfParam-class), 24  
 ScanVcfParam,missing-method  
     (ScanVcfParam-class), 24  
 ScanVcfParam-class, 24  
 select,PolyPhenDb-method  
     (PolyPhenDb-class), 8  
 select,SIFTDb-method (SIFTDb-class), 26  
 seqlevels,VCF-method (VCF-class), 36  
 show,AllVariants-method  
     (VariantType-class), 35  
 show,PromoterVariants-method  
     (VariantType-class), 35  
 show,VAFilter-method (VAFilter-class), 31  
 show,VAFilterResult-method  
     (VAFilterResult-class), 33  
 show,VariantType-method  
     (VariantType-class), 35  
 show,VCF-method (VCF-class), 36  
 show,VCFHeader-method  
     (VCFHeader-class), 39  
 SIFT (SIFTDb-class), 26  
 SIFTDb (SIFTDb-class), 26  
 SIFTDb-class, 26  
 SIFTDbColumns, 28  
 SimpleList, 36, 39  
 SnpMatrix, 6, 7  
 SpliceSiteVariants (VariantType-class), 35  
 SpliceSiteVariants-class  
     (VariantType-class), 35  
 stats (VAFilterResult-class), 33  
 stats,VAFilterResult-method  
     (VAFilterResult-class), 33  
 strand,VCF-method (VCF-class), 36  
 strand<-,VCF-method (VCF-class), 36  
 SummarizedExperiment, 36, 37, 39

summarizeVariants, 5, 29, 30  
 summarizeVariants,GRangesList,VCF,function-method  
     (summarizeVariants), 29  
 summarizeVariants,GRangesList,VCF,VariantType-method  
     (summarizeVariants), 29  
 summarizeVariants,TranscriptDb,VCF,CodingVariants-method  
     (summarizeVariants), 29  
 summarizeVariants,TranscriptDb,VCF,FiveUTRVariants-meth  
     (summarizeVariants), 29  
 summarizeVariants,TranscriptDb,VCF,IntronVariants-method  
     (summarizeVariants), 29  
 summarizeVariants,TranscriptDb,VCF,PromoterVariants-meth  
     (summarizeVariants), 29  
 summarizeVariants,TranscriptDb,VCF,SpliceSiteVariants-meth  
     (summarizeVariants), 29  
 summarizeVariants,TranscriptDb,VCF,ThreeUTRVariants-met  
     (summarizeVariants), 29  
  
 TabixFile, 15, 17, 19, 22–24  
 ThreeUTRVariants (VariantType-class), 35  
 ThreeUTRVariants-class  
     (VariantType-class), 35  
 TranscriptDb, 3, 12, 21, 29, 31  
 transcriptLocs2refLocs, 21  
  
 upstream (VariantType-class), 35  
 upstream,AllVariants-method  
     (VariantType-class), 35  
 upstream,PromoterVariants-method  
     (VariantType-class), 35  
 upstream<- (VariantType-class), 35  
 upstream<-,AllVariants-method  
     (VariantType-class), 35  
 upstream<-,PromoterVariants-method  
     (VariantType-class), 35  
  
 VAFilter, 31, 34  
 VAFilter (VAFilter-class), 31  
 vaFilter, 34  
 vaFilter (VAFilter-class), 31  
 vaFilter,function-method (VAFilter-class),  
     31  
 vaFilter,missing-method (VAFilter-class),  
     31  
 vaFilter,VAFilter-method (VAFilter-class),  
     31  
 VAFilter-class, 31  
 VAFilterResult, 31–34  
 VAFilterResult (VAFilterResult-class), 33  
 VAFilterResult-class, 33  
 VariantType-class, 35  
 VAUtil-class, 36  
 VCF, 3, 7, 12, 16, 29, 41

VCF (VCF-class), [36](#)  
VCF-class, [36](#)  
vcfFixed (ScanVcfParam-class), [24](#)  
vcfFixed<- (ScanVcfParam-class), [24](#)  
vcfGeno (ScanVcfParam-class), [24](#)  
vcfGeno<- (ScanVcfParam-class), [24](#)  
VCFHeader (VCFHeader-class), [39](#)  
VCFHeader-class, [39](#)  
vcfInfo (ScanVcfParam-class), [24](#)  
vcfInfo<- (ScanVcfParam-class), [24](#)  
vcfTrimEmpty (ScanVcfParam-class), [24](#)  
vcfTrimEmpty<- (ScanVcfParam-class), [24](#)  
vcfWhich (ScanVcfParam-class), [24](#)  
vcfWhich<- (ScanVcfParam-class), [24](#)  
  
writeVcf, [39](#), [41](#)  
writeVcf,VCF,character-method (writeVcf),  
[41](#)